**KWIC System Architecture with Abstract Data Types and Implicit Invocation**

The KWIC (Key Word In Context) system is designed using an **Abstract Data Type (ADT) Architecture** and follows the **Implicit Invocation** design pattern. This architecture ensures that components are modular, loosely coupled, and able to trigger behavior without explicit method calls. Below is an updated breakdown of the KWIC system based on the provided code.

**1. Abstract Data Type (ADT) Architecture**

The system is organized around ADTs, where each component is responsible for managing its own data and operations. The components only interact through clearly defined interfaces, ensuring modularity and maintainability. The primary components include:

* **LineStorage**: Manages storage and retrieval of lines and circular shifts.
* **CircularShift**: Handles the generation of all possible circular shifts for a given line.
* **Alphabetizer**: Sorts the circular shifts alphabetically and merges them with the existing sorted shifts.
* **InputModule**: Reads input lines and passes them to LineStorage.
* **Output**: Displays or retrieves the results of the sorted KWIC index.

**2. Implicit Invocation**

The system is designed with implicit invocation, where components communicate indirectly. For example, the Master class implicitly invokes the sorting process in Alphabetizer when a new line is processed. This design decouples components, allowing them to operate independently and making the system more flexible and scalable.

**KWIC Components, Roles, and Connections**

1. **Master Class**
   * **Role**: Acts as the central coordinator of the KWIC system. It manages interactions between the various components and orchestrates the entire process, from input handling to output generation.
   * **Components**:
     + LineStorage: Stores input lines and sorted circular shifts.
     + InputModule: Reads input lines and stores them in LineStorage.
     + CircularShift: Generates circular shifts for the lines stored in LineStorage.
     + Alphabetizer: Sorts and merges the circular shifts in LineStorage.
     + Output: Retrieves and prints the sorted KWIC results.
   * **Key Method**:
     + process\_line: Manages the process of reading a line, generating circular shifts, alphabetizing them, and updating the KWIC index.
2. **InputModule Class**
   * **Role**: Handles input, reading each line and passing it to LineStorage.
   * **Key Method**:
     + read: Reads a single line of input and stores it in LineStorage.
3. **LineStorage Class**
   * **Role**: Central data store for lines and their circular shifts. It manages three arrays: lines, sorted\_shifts, and temp\_shifts.
   * **Attributes**:
     + lines: List of input lines.
     + sorted\_shifts: List of sorted circular shifts.
     + temp\_shifts: List of circular shifts generated from a line.
   * **Key Methods**:
     + setline: Adds a new line to the specified array (lines, sorted\_shifts, or temp\_shifts).
     + getline: Retrieves a specific line or shift based on its index.
     + word: Returns the number of words in a specific line.
     + update\_array: Updates the array with a new set of shifts or lines.
     + getArray: Retrieves a specified array.
4. **CircularShift Class**
   * **Role**: Generates all possible circular shifts for a given input line.
   * **Key Methods**:
     + setup: Initializes the process of generating circular shifts for the most recently stored line in LineStorage.
     + shiftWords: Produces circular shifts for the words in a line.
     + CSLine: Retrieves a specific circular shift from the temporary shifts array.
5. **Alphabetizer Class**
   * **Role**: Sorts the generated circular shifts lexicographically and merges them with any existing sorted shifts in LineStorage.
   * **Key Methods**:
     + alpha: Sorts the circular shifts alphabetically and merges them with the previously stored shifts.
     + ith: Retrieves the i-th sorted circular shift.
6. **Output Class**
   * **Role**: Responsible for displaying or retrieving the results of the KWIC system.
   * **Key Methods**:
     + print\_all\_KWIC: Prints all sorted circular shifts stored in LineStorage.
     + print\_KWIC: Prints the KWIC shift at a specific index.

**KWIC Process Flow**

1. **Input (InputModule)**:
   * A line of text is passed to the system.
   * InputModule reads the line and stores it in LineStorage.
2. **Storage (LineStorage)**:
   * The line is stored in the lines list of LineStorage.
3. **Circular Shifting (CircularShift)**:
   * CircularShift generates all circular shifts of the input line.
     + For example, for the line "hello world":
       - The shifts are: "hello world" and "world hello".
4. **Sorting (Alphabetizer)**:
   * Alphabetizer sorts the shifts lexicographically.
   * The sorted shifts are merged into LineStorage’s sorted\_shifts array.
5. **Output (Output)**:
   * The Output class prints or retrieves the sorted KWIC lines.

**Connections Between Components**

* **Master: Manages the overall data flow between components.**
* **InputModule → LineStorage: Stores input lines.**
* **LineStorage → CircularShift: Provides input lines for circular shifting.**
* **CircularShift → Alphabetizer: Passes generated shifts for sorting.**
* **Alphabetizer → LineStorage: Updates LineStorage with sorted shifts.**
* **Output → Alphabetizer: Retrieves sorted shifts for display.**

**Constraints for the KWIC System**

1. **Modularity: Each component operates independently, ensuring the system is easier to maintain.**
2. **Encapsulation: Internal data structures are hidden from other components, which can only access exposed methods.**
3. **Separation of Concerns: Each component handles a distinct part of the process (input, storage, shifting, sorting, output).**
4. **Efficiency: Sorting and merging are handled incrementally to avoid reprocessing the entire index.**
5. **Loose Coupling: The system relies on implicit invocation, reducing dependencies between components.**

**Key Design Choices**

* **Separation of Concerns: Input, circular shifting, sorting, and output are handled by separate components.**
* **Incremental Processing: Each line is processed individually, allowing for continuous updates to the KWIC index.**
* **Efficient Sorting: The system efficiently merges new shifts with existing sorted shifts in LineStorage.**